ELSEVIER

Theoretical Computer Science 145 (1995) 27–43

Theoretical Computer Science

Optimal parallel algorithms for path problems on planar graphs

G. Srikrishna, C. Pandu Rangan*

Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India

Received September 1991; revised July 1994
Communicated by M. Nivat

Abstract

Given a biconnected planar graph G and a pair of vertices s and t, the two disjoint problem asks to find a pair of internally disjoint paths from s to t. We present a simple and efficient parallel algorithm for the same. Our algorithm uses the notion of bridges in a novel way and this results in a more elegant and simple algorithm than the existing one. The all-bidirectional-edges (ABE) problem is to find an edge labeling such that an edge (u, v) in E is labeled $\langle u, v\rangle$ or $\langle v, u\rangle$ or both depending on the existence of a simple path from s to t that visits the vertices in the order u, v or v, u or both, respectively. We present an optimal parallel algorithm for the same.

1. Introduction

Let $G = (V, E)$ be a finite undirected graph. Let $|V| = n$ and $|E| = m$. The two-path problem (TPP) is defined as follows. Given two pairs of vertices s, t and u, v in V, find two paths, one from s to t to another from u to v such that, the paths are *vertex disjoint* (here after we simply call vertex disjoint as disjoint). We assume, without loss of generality that, neither s, t nor u, v are adjacent. This problem has obvious applications in certain routing situations and has been studied well from point of view of sequential computation [18, 22, 21, 17]. The current best sequential algorithm for TPP on arbitrary undirected graphs runs in $O(nm)$ time [22, 21]. However, TPP on directed graphs is shown to be NP-complete in [3]. A more general problem of deciding whether there exist k *pairwise disjoint paths* between s_i and t_i for k given pairs of vertices $\{s_1, t_1\}, \ldots, \{s_i, t_i\}$ is known to be NP-complete when k is a variable [8] and it remains NP-complete even if G is constrained to be planar [14]. However, if k is fixed the problem is more tractable. In [9] an $O(n^2m)$ algorithm is given for this

* Corresponding author. E-mail: rangan@iitm.ernet.in

SSDI 0304-3975(94)00189-8

problem. However, no practical algorithm is known even for the case when $k = 3$. The complexity of the algorithm in [9] involves ludicrously large constants. Let s, t be any two distinguished vertices in G. The all-bidirectional-edges (ABE) problem is to find an edge labeling such that an edge (u, v) in E is labeled $\langle u, v\rangle$ or $\langle v, u\rangle$ or both depending on the existence of a (simple) path from s to t that visits the vertices in the order u, v or v, u or both, respectively. In [16] an $O(mn)$ algorithm is given for this problem by partitioning the graph into sets of paths and bridges and analyzing them. This problem arises in the context of simulation of MOS transistor network [16].

The parallel random-access machine (PRAM) has emerged in the recent past as a successful model for the design and analysis of parallel algorithms, the main advantage being the ability to express parallelism without communication concerns [12, 7]. An optimal parallel algorithm is a parallel algorithm for which the work (parallel time multiplied by the number of processors used) performed by that algorithm in the worst-case is within a constant factor of the time complexity of the best-known sequential algorithm or the optimal sequential algorithm. Since the definition of an optimal parallel algorithm does not capture the speed of an algorithm, an additional goal is frequently stipulated: minimize the running time while not violating the condition of optimality. Typically, a running time which is a poly-logarithm function of the length of the input is considered good. Now, the challenge is to invent techniques that will help realize these goals.

In this paper, we give parallel algorithms for TPP on planar graphs. TPP for a general graph is generally solved first by dividing the graph into triconnected components and solving TPP on suitable triconnected components and combining the solutions to obtain the required paths in the input graph. In [4] an *almost* optimal parallel algorithm is given for dividing the graph into triconnected components on a CRCW PRAM. This algorithm is long and complicated and the two problems that preclude this algorithm from achieving optimality are the integer sorting and finding spanning tree. However, when restricted to planar graphs the spanning tree can be found optimally [6] while the parallel integer sorting is still suboptimal though can be done in $O(\log n)$ time (see [5] for details). In [10], TPP was solved on a planar graph on CRCW PRAM as follows. Divide the graph into triconnected components using the algorithm given in [4] and solve TPP on suitable triconnected components. Now, the problem is reduced to that of solving TPP on a triconnected planar graph. In a 3-connected planar graph, three vertex disjoint paths from s to t and three vertex disjoint paths from u to v are constructed. Then, complicated operations are performed on these paths and the two disjoint paths are obtained whenever they exist. The algorithm runs in $O(\log n)$ time using $n \propto (n, n)/\log n$ processors. Thus, the spirit of their algorithm is similar to that of [22]. In this paper, we give an $O(\log n)$ time algorithm on CRCW PRAM for TPP on planar graphs without dividing the graph into triconnected components. The only algorithm the precludes our algorithm from achieving optimality is suboptimal parallel integer sorting (see [5] for details). The algorithm exploits many structural properties of bridges of planar graphs and our

approach gives a new insight into application of bridges for the planar graphs. Thus, the algorithm is not similar to that of [10]. The algorithm is elegant and simple and can be easily implemented. A new sequential algorithm for TPP on planar graphs can be easily derived from this algorithm.

In [19] a linear algorithm is given for ABE problem on planar graphs. The spirit of their algorithm is same as that of the algorithm given in [16], but exploits some structural properties of bridges of planar graphs to reduce the complexity. As a by-product of parallel computation on planar graphs using bridges, we could solve some key subproblems employed in the algorithm for ABE problem for planar graph presented in [19]. Thus, we present an $O(\log n)$ time $O(n/\log n)$ processor algorithm on CRCW PRAM for ABE problem.

2. Basic definitions

Let $G = (V, E)$ be a graph. For all graph theoretic terms not explicitly defined in this paper see [2]. We denote the subgraph induced by $V' \subset V$, by $G' = \langle V' \rangle$.

Definition 2.1. A graph G is said to be *planar* if there exists some geometric representation of G which can be drawn on a plane such that edges intersect only at their end vertices. Such a drawing is called *planar embedding* of G. A planar embedding partitions the plane into a number of connected regions; the closures of these regions are called the *faces* of the embedding.

If P is a path between any two vertices a, b then, the whole path will be denoted by $P[a, b]$. If c, d are any two vertices on P then $P[c, d]$ denotes the subpath of P from c to d including both c and d; $P[c, d[$ denotes the subpath including c but excluding d and $P]c, d[$ denotes the subpath excluding both c and d. Also by $P[c, d]$ we mean the subpath from c to d along P (in $P[d, c]$ edges are traversed in the opposite direction as compared to $P[c, d]$). If P and P' are paths, respectively, from a to b and b to c and if P and P' are vertex disjoint except for b then the *concatenation* of P and P' is denoted by $P[a, b] * P'[b, c]$.

Definition 2.2. A connected graph is said to have a separation vertex v (also called an articulation point or cut vertex) if there exist vertices a and b, $a \neq v$ and $b \neq v$, such that all the paths from a to b pass through v. A graph which has a separation vertex is called separable and one which has none is called nonseparable (also called biconnected). Let $V' \subset V$. The induced graph $G' = \langle V' \rangle$ is called a *nonseparable component* (also called biconnected component) if G' is nonseparable and if for every larger $V'' \supset V$, the induced graph $G'' = \langle V'' \rangle$ is separable.

Definition 2.3. Given a connected graph G, its *bc-tree*, denoted by $T(G)$, is defined as follows. $T(G)$ is a bipartite graph with bipartition $\langle B, S \rangle$ where, there is a 1–1

mapping f of the biconnected components of G onto the vertices of B and a 1–1 mapping g of the separation vertices of G onto the vertices of S. A vertex $f(C)$ of B is adjacent to a vertex $g(x)$ of S iff the biconnected component C contains the separation vertex x. Vertices of B are referred to as *b-vertices* and those of S as *c-vertices*. In $T(G)$ any path contains B and S vertices alternatively.

Let s and t be any two distinguished vertices of G, and let C_s and C_t be the biconnected components containing s and t, respectively. Let $P = f(C_1), g(a_1), f(C_2), g(a_2), \ldots, f(C_{k-1}), g(a_{k-1}), f(C_k)$, where $C_1 (= C_s), C_2, \ldots, C_k (= C_t)$ are the biconnected components of G and $a_1, a_2, \ldots, a_{k-1}$ are the separation vertices of G such that a_i separates c_i and c_{i+1} for all i, $1 \leqslant i \leqslant k-1$. Then, P is called a *chain graph.*

Definition 2.4. A connected graph is said to have a separation pair u, v, if there exist a pair of vertices a and b, $a \notin \{u, v\}$ and $b \notin \{u, v\}$, such that all the paths from a to b pass through either u or v. A graph which has no separation pair is called a triconnected graph. A triconnected component is similarly defined.

Definition 2.5. Let G_1 be a subgraph of G. A vertex of attachment of G_1 in G is a vertex of G_1 that is incident in G with some edge not belonging to G_1.

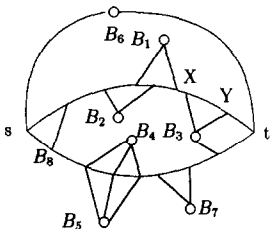
Definition 2.6 (Tutte [24]). Let J be a subgraph of G. Let G' be the graph derived from G by deleting the vertices of J and all their incident edges. Let C be any connected component of G'. Let B be the subgraph obtained from C by adjoining to it the edges of J that are incident on C. The subgraph B is called a *proper* bridge. The component C of G is called the *nucleus* of B.

The set of vertices of attachment of a bridge B of a subgraph J in G is denoted by $W(G, B) = \{v_0, v_1, \ldots, v_{k-1}\}$. $w(G, B)$ stands for $|W(G, B)|$. If $w(G, B) = k$ then B is said to be a *k-bridge.*

Definition 2.7. Let G and J be as in Definition 2.6. An edge $e = (u, v)$ of G not belonging to J but having both of its ends in J is called a *degenerate bridge.*

Remark. In the rest of the paper, we will assume that the subgraph J of G is either a cycle or a path in G. In Fig. 1, we give an example of bridges of a cycle J passing through s and t. In this example, B_1 and B_2 are proper bridges and B_8 is a degenerate bridge.

Definition 2.8. Let the set of vertices of attachment of a bridge B of a cycle J in G be $W(G, B) = \{v_0, v_1, \ldots, v_{k-1}\}$ and let $v_0, v_1, \ldots, v_{k-1}$ be their enumeration in their cyclic order on J. The vertices of attachment dissect J into k paths $L_0, L_1, \ldots, L_{k-1}$, where $L_j = J[v_j, v_{j+1 \,(\mathrm{mod}\, k)}]$. These paths are called *residual* paths.



X = $s_P(B_3)$
Y = $t_P(B_3)$
B_1,B_2 are P-bridges
B_4,B_5,B_7 are Q-bridges
B_3 is a PQ-bridge
B_6 is an st-bridge
B_1 and B_2 interlace
B_4 and B_5 interlace
B_4 and B_7 do not interlace

Fig. 1. A cycle and its bridges.

Definition 2.9. Let B and B' be two distinct bridges of a cycle J of G. We say that B avoids B' iff one of the two following conditions is satisfied:

(1) $w(G, B) = 1$ or $w(G, B') = 1$.

(2) All the vertices of attachment of B are contained in a single residual path L of B'.

If B and B' do not avoid one another we say that they overlap.

If there are two vertices of attachment a and b of B and two vertices of attachment c and d of B', all four distinct, such that a and b separate c and d in J, then we say that the two bridges interlace.

Notice that, if B and B' are bridges where $W(B, G) = W(B', G)$ and $W(B, G) = 3$, then B and B' interlace but they do not overlap.

If the two bridges have exactly the same vertices of attachment we say they are *equivalent*. These definitions extend verbatim if the subgraph J is a path.

Remark. In Fig. 1, bridges B_1 and B_2 interlace and bridges B_4 and B_5 interlace. The bridge B_1 avoids B_3. Also notice that the bridge B_7 avoids every bridge.

Definition 2.10. In the rest of the paper we follow the following notations. Let G be an undirected biconnected graph with two distinguished vertices s and t with two internally disjoint paths $P[s, t]$ and $Q[s, t]$. Now consider the cycle $C = P \cup Q$ formed by P and Q. P and Q are called the *complementary paths* of C. The bridges of C are classified as follows.

P-bridge: A bridge with at least one vertex of attachment in $P\,]s, t[$ and none in Q is called as P-bridge.

Q-bridge: A bridge with at least one vertex of attachment in $Q\,]s, t[$ and none in P is called as Q-bridge.

PQ-bridge: A bridge with at least one vertex of attachment in $P\,]s, t[$ and at least one vertex of attachment in $Q\,]s, t[$ is called PQ-bridge.

s–t-bridge: A bridge with s and t as only vertices of attachment is called an s–t-bridge.

Definition 2.11. A cycle C in G (passing through two vertices s and t) in which every P or Q-bridge avoids every PQ-bridge is called an *s–t-ambitus*.

Remark. In Fig. 1, B_1, and B_2 and P-bridges. B_4, B_5 and B_7 are Q-bridges and B_6 is an s–t-bridge. B_3 and B_8 are PQ-bridges.

Definition 2.12. Let B be any bridge of C. Let a and b be any two vertices of attachment of B. A cross-cut from a to b through B is a path from a to b such that none of its internal vertices belong to C and all of its edges are in B. Such a cross-cut is denoted by $CC[a, b]$. The vertices a, b are called end vertices of cross-cut. A P cross-cut is one whose both end vertices are in P. Q and PQ cross-cuts are similarly defined. To emphasize the fact that a cross-cut is either P or Q or PQ-cross-cut we may add subscripts P or Q or PQ. Interlacing, avoiding, and overlapping cross-cuts can be similarly defined.

Definition 2.13. For any bridge B of C, $s_P(B)$ denotes the vertex of attachment of B in P which is nearest along P to s and $t_P(B)$ denotes the vertex of attachment of B in P which is nearest along P to t if such vertices exist. Similarly $s_Q(B)$ and $t_Q(B)$ are defined whenever they exist. $s_P(B)(s_Q(B))$ is called left-end vertex and $t_P(B)(t_Q(B))$ is called right-end vertex of B.

Definition 2.14. A vertex a on P is said to be *under* a bridge B if it is in $P[s_P(B), t_P(B)]$. It is said to be *properly under* a bridge B if it is in $P]s_P(B), t_P(B)[$. Similar definitions can be given for vertices being under or strictly under a cross-cut.

Definition 2.15. A vertex a in P is said to be to the left of another vertex b in P if a is in $P[s, b]$ and is strictly to the left of b if a is in $P[s, b[$. The notion of *right* and *strictly to the right* can be defined analogously. Similar definitions can be given for the vertices in Q.

Definition 2.16. In G, let S be a nonempty set of bridges of the cycle C. A block B of S is a nonempty subset of S, such that,

(1) every bridge of S that overlaps with a bridge of B is also in B, and

(2) no nonempty proper subset of B satisfies condition (1).

Note that the set of blocks as defined above induces a partition on S. By *a block of* S, we mean *a block of bridges in* S where S is the set of all bridges of C in G. If all bridges of a block are either P-bridge, Q-bridge or PQ-bridge then they are referred to as block of P, Q or PQ bridges, respectively.

Definition 2.17. Let a and b be two vertices on $P(Q)$ and a is strictly to the left of b. Let there be two vertices c, d in $Q(P)$ and c is strictly to the left of d. If there is a cross-cut between a and d, through a bridge B, disjoint from a cross-cut between b and c, through a bridge B' then, a and b are said to cross-over. The cross-cut between a and d is said to use B and the one between b and c is said to use B'.

3. Algorithmic details

Now, we give some reductions which reduce the TPP and ABE problems on an arbitrary planar graph $G = (V, E)$ to those on a biconnected planar graph. In the rest of the paper we assume, without loss of generality, that the graph is connected unless otherwise specified. Let $|V| = n$ and $|E| = m$.

Lemma 1. *ABE problem on an arbitrary planar graph G is reducible to that on a biconnected planar graph in* $O(\log n)$ *time using* $O(n/\log n)$ *processors.*

Proof. Let $(C_s =)C_1, C_2, \ldots, C_k(= C_t)$ be the biconnected components corresponding to b-vertices in the chain graph of G containing s and t. Notice that edges in other biconnected components are not contained in any path between s and t. Hence, all the other biconnected components can be removed from G. Further, the ABE problem can be solved on each of the biconnected components independently. All biconnected components of a planar graph can be found optimally in $O(\log n)$ time on a CRCW PRAM [6]. A path from s to t can be found by the algorithm of [13] within the same processor time bounds. □

Lemma 2. *TPP on an arbitrary graph is reducible to that on a biconnected graph in* $O(\log n)$ *time and* $O(n/\log n)$ *processors.*

Proof. Let $(C_s =)C_1, C_2, \ldots, C_k(= C_t)$ be the biconnected components corresponding to the b-vertices of the chain graph containing s and t. If u or v does not lie in any of these biconnected components, TPP is trivially solved and if they lie in two different components of the chain then, the two paths does not exist. Hence, without loss of generality, we assume that, one of u, v lie in one of these biconnected components or both u, v lie in a single biconnected component. Let u be the vertex lying in a biconnected component say, C_i. If v is in C_i then, solving TPP between s, t and u, v is equivalent to that of solving the same between the pairs of vertices u, v and v_{i-1}, v_i else is between v_{i-1}, v_i and u, w where v_i, is the cut vertex between the components C_{i-1} and C_i (v_{i-1} is similarly defined) and w is the first cut vertex in the other disjoint path between u and v. Since biconnected components and a path from s to t can be found optimally in $O(\log n)$ time [6, 13], the overall reduction can be done within same processor-time bounds. □

Hence, in the rest of the paper we assume G to be biconnected and planar and we use only CRCW PRAM unless otherwise specified.

3.1. Some preliminary results

We need to find bridges of a cycle in G for all our algorithms and therefore we outline how to compute bridges in $O(\log n)$ time using only $O(n/\log n)$ processors.

Step 1: Find two disjoint paths, P and Q between s and t and form the cycle $C = P \cup Q$.

Step 2: Delete C from G and find connected components.

Step 3: Label the bridges as P, Q or PQ, depending on whether their vertices of attachment are in, respectively, P, Q or both.

Step 4: END.

The two complementary paths can be found optimally in $O(\log n)$ time on CRCW PRAM by the algorithm of [13]. The two steps above can be trivially implemented within the same processor-time bounds using [6] and standard techniques.

Definition 3.1. Let s and t be two vertices in a biconnected graph G. Let P and Q be two internally disjoint paths from s to t. Then the cycle $C = P \cup Q$ is an s–t-ambitus if no P or Q bridge interlaces with a PQ-bridge.

The ambitus of a planar biconnected graph can be found in $O(\log n)$ time using $O(n/\log n)$ processors on EREW PRAM [11]. As will be seen shortly, ambitus plays a pivotal role in the solutions of disjoint path problem and ABE problem.

3.2. *Algorithmic details of TPP*

Hereafter, we assume that the cycle C is an s–t-ambitus and P and Q as two complementary paths from s to t. In the discussion to follow we denote the two disjoint paths, one from s to t and another from u to v, respectively, by $D[s, t]$ and $D[u, v]$ and set a Boolean flag, TPP to TRUE iff both $D[s, t]$ and $D[u, v]$ exist. Also, in the discussion to follow, as a degenerate case, we assume that, if u, v lie in $P(Q)$, they are considered to lie in $P(Q)$-bridge unless otherwise specified. In fact, we show later that, both cases are equivalent. Now we make the following observation regarding s–t-bridges.

Observation 1. *If there is at least one s–t-bridge in G then, TPP is TRUE iff*

(1) *both u and v are in (same or different) P, Q or PQ bridges, or*
(2) *both u and v are in the same s–t-bridge, or*
(3) *u is in a P-bridge and v is in a Q-bridge (or vice versa) and there is at least one PQ-bridge, or*
(4) *u is in a $P(Q)$-bridge and v is in a PQ-bridge (or vice versa).*

Proof. If any of the above conditions is satisfied then, take $D[s, t]$ as a path through an s–t-bridge and use the remaining graph for finding $D[u, v]$. For "only if" part, observe the only possibility is that u is in an s–t-bridge and v is in P or Q or PQ or in another s–t-bridge or in paths P or Q (or vice versa). Since $\{s, t\}$ forms a separation pair for u and v in all these cases, TPP is FALSE. □

These conditions can be easily checked and paths can be found in $O(\log n)$ time using $O(n/\log n)$ processors [13]. Henceforth, without loss of generality, we assume

that no s–t-bridges are present. Following are possible cases depending on the position of u. Notice there is a symmetric case if we replace u by v. The case corresponding to v will not be discussed here.

Case 1: u is in the nucleus of a PQ-bridge.

(1a) v is in the nucleus of a PQ-bridge, either same or different.

In this case TPP is TRUE. Take P as the path $D[s, t]$ and use a subpath of Q for $D[u, v]$ if necessary.

(1b) v is in nucleus of a $P(Q)$-bridge either same or different.

In this case TPP is TRUE. Take $Q(P)$ as $D[s, t]$ and use a subpath of $P(Q)$ for $D[u, v]$ if necessary.

Case 2: u is in the nucleus of a P-bridge.

(2a) v is in the nucleus of a P-bridge, either same or different.

In this case TPP is TRUE. Take Q as $D[s, t]$ and use subpath of P for $D[u, v]$ if necessary.

(2b) v is the nucleus of a Q-bridge.

This is the only nontrivial case and rest of the section is devoted to this case.

Lemma 3. *Let u be in a P-bridge A and v in a Q-bridge B. There are two disjoint paths $D[s, t]$ and $D[u, v]$ iff there is at least one path P' with one of its end vertices in $\{s_P(A), t_P(A)\}$ and the other in $\{s_Q(B), t_Q(B)\}$ such that P is disjoint with $D[s, t]$.*

Proof. The *if* part is easy to show by explicitly constructing the paths. So, we proceed to show the *only if* part. Notice that by definition there is exactly one $P(Q)$-bridge in any block of $P(Q)$-bridges in s–t-ambitus. Since every PQ-bridge avoids every $P(Q)$-bridge, either s_P or t_P is in $D[u, v]$ depending on whether, path $D[u, v]$ comes out of s_P or t_P. By similar argument, we can show that either s_Q or t_Q is also present in $D[u, v]$. Hence, if $D[u, v]$ is disjoint from $D[s, t]$ then, there is at least one path with one of its end vertices in $\{s_P(A), t_P(A)\}$ and another in $\{s_Q(B), t_Q(B)\}$. □

Corollary 1. *If either $D[s, t]$ or $D[u, v]$ has at least one vertex in a $P(Q)$-bridge B then, the other path cannot have a vertex in B or $P[s_{P(Q)}(B), t_{P(Q)}(B)]$.*

Proof. In the above proof we notice that any path that contains a vertex from $P(Q)$-bridge has $s_{P(Q)}(B)$ or $t_{P(Q)}(B)$. Since the paths are disjoint, the corollary is proved. □

Corollary 2. *Neither of the two disjoint paths need to use nucleus of a $P(Q)$-bridge B unless, v is in the nucleus of a $P(Q)$-bridge.*

Proof. From the above corollary, we see that we can use $P[s_{P(Q)}(B), t_{P(Q)}(B)]$ instead of nucleus of the $P(Q)$-bridge. □

From the above lemmas and corollaries its obvious that, the graph G can be assumed, without loss of generality, to contain the cycle (ambitus) C with only PQ-bridges and u, v lie in P, Q, respectively.

Lemma 4. *Let a, b, c, d be four vertices on C. Let a and b be on P and a is strictly to the left of b. Let c and d be on Q and c is strictly to the left of d. Then, no single PQ-bridge can contain both the disjoint paths, one from a to d and another from b to c.*

Proof. The proof is by contradiction. Let B be a bridge containing the disjoint paths, one from a to d and another from b to c. Without loss of generality assume that B is in the interior face bounded by C. Then the cycle $C' = P[s, a] * CC[a, d] * Q[d, s]$, where $CC[a, d]$ is cross-cut through B from a to d, is a Jordan curve. Now, if there is a path from b to c lying in the interior region bounded by C, then form a subdivision of this path, if necessary, such that there is a vertex v inside C'. This, by Jordan curve theorem, contradicts the planarity of G. Also, note that the path cannot be in the exterior face because B is embedded inside. □

Thus, it is clear that no two subpaths, one of $D[s, t]$ and another of $D[u, v]$ are in a single PQ-bridge. They have to be *alternating* between PQ-bridges. Thus, it is trivial to see that TPP is FALSE if there is exactly one PQ-bridge for C.

Now we give necessary and sufficient conditions for TPP to be TRUE.

Theorem 1. *TPP is TRUE iff*

(1) *there exists a pair of distinct vertices a, b on P such that a is strictly to the left of b as well as u and a cross-cut with a as one end vertex uses a PQ-bridge B_1 such that $t_Q(B_1)$ is strictly to the right of v and a cross-cut with b as one end vertex uses B_1 (B_2 different from B_1) such that a and b cross-over* (Fig. 2(a)), or

(1a) *there exists a pair of vertices a, b on Q such that a is strictly to left of b as well as v and the cross-cut with a as one end vertex uses a PQ-bridge B_1 such that $t_P(B_1)$ is strictly to the right of u and the cross-cut with b as one end vertex uses B_2 (B_2 different from B_1) such that a and b cross-over* (Fig. 2(b)), or

(2) *u is strictly under a PQ-bridge B_1 and there is another PQ-bridge that has a vertex of attachment in $P]s_P(B_1), t_P(B_1)[$, or*

(2a) *u is strictly under a PQ-bridge B_1 and there is another PQ-bridge that has a vertex of attachment in $P]s_Q(B_1), t_Q(B_1)[$.*

Proof. It can be seen that conditions (1) and (1a) are symmetric and so are (2) and (2a). Hence only conditions (1) and (2) are proved here.

If part: If condition (1) is TRUE then, $D[s, t]$ is $P[s, a] * CC_1[a, t_Q(B_1)] * Q[t_Q(B_1), t]$ and $D[u, v]$ is $P[u, b] * CC_2[b, t_Q(B_2)] * Q[t_Q(B_2), v]$, where CC_1 and CC_2 are cross-cuts through the bridges B_1 and B_2, respectively. If condition (2) is TRUE, then let the vertex of attachment of B_2 is $P]s_P(B_1, t_P(B_1)[$ be c. Then the two

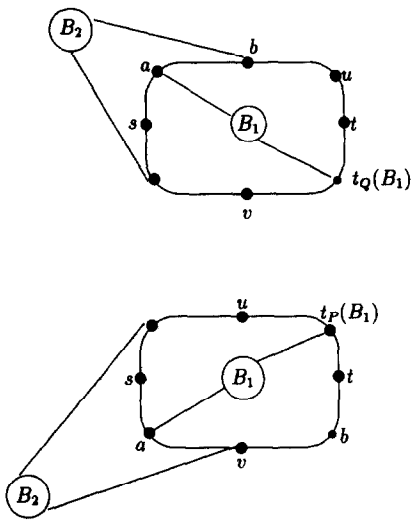


Fig. 2(a) and 2(b).

disjoint paths are $D[s, t] = P[s, s_P(B_1)] * CC_1[s_P(B_1), t_P(B_1)] * P[t_P(B_1), t]$ and $D[u, v] = P[u, c] * CC_2[c, d] * Q[d, v]$, where d is any vertex of attachment of B_2 on Q.

Only if part: The proof is by contradiction. Assume there are two disjoint paths $D[s, t]$ and $D[u, v]$ without satisfying conditions (1) and (2). Since u, v are respectively in P, Q, there is a PQ-cross-cut in $D[u, v]$. Also, there is another cross-cut in $D[s, t]$ as $D[s, t]$ is disjoint from $D[u, v]$. Further, by Lemma 4 each of the cross-cuts use different bridges. To make things easy make the following changes to the paths. If there is a $P(Q)$-cross-cut in a path, say from a to b and if the other path does not have any vertex in $P(Q)]a, b[$ then replace that cross-cut by $P(Q)[a, b]$. It can be seen that these new paths are disjoint if original paths are disjoint. In the discussion to follow, we call them as $D[s, t]$ and $D[u, v]$. Following are the possible cases depending on the positions of u and v.

Case 1: u is strictly under a P-cross-cut of $D[s, t]$ or v is strictly under a Q-cross-cut of $D[s, t]$.

Case 2: u is not under a P-cross-cut of $D[s, t]$ and v is not under a Q-cross-cut of $D[s, t]$.

Case 1: Without loss of generality, assume that u is under a P-cross-cut. Let the cross-cut be $CC_{P1}[a, b]$. By Lemma 4 there cannot be another cross-cut disjoint with $CC_{P1}[a, b]$ such that both these cross-cuts are in the same PQ-bridge. Therefore, if there is another path from u to v disjoint with that from s to t, then there are two vertices c and d such that c is strictly to the right of a and strictly to the left of b (on P) and d on Q such that $CC_{P2}[c, d]$ and $CC_{P1}[a, b]$ interlace. Further, these cross-cuts are in different bridges and hence condition (2) is satisfied – a contradiction.

Case 2: In this case there is a PQ-cross-cut

(1) either from a vertex strictly to the left of u to a vertex strictly to the right of v,

(2) or from a vertex strictly to the left of v to a vertex strictly to the right of u.

Without loss of generality assume that $D[s, t]$ contains a cross-cut from a vertex which is strictly to the left of u to a vertex strictly to the right of v. Consider the set CC_{PQ} of all the cross-cuts that satisfy this condition (in fact, this relation induces a partition on the set of all cross-cuts of G). Let $CC_{st}[a, b]$ be a cross-cut in CC_{PQ} of $D[s, t]$. Now, there is another cross-cut $CC_{uv}[c, d]$ belonging to CC_{PQ}, of $D[u, v]$ such that these cross-cuts interlace (by Lemma 4). If $CC_{uv}[c, d]$ is a PQ-cross-cut then condition (2) is satisfied – a contradiction. So assume that it is a P-cross-cut. Consider the set CC_P defined as follows:

(1) $CC_{uv}[c, d]$ is in CC_P.

(2) Any cross-cut that interlaces with a cross-cut in CC_P is in CC_P.

(This is similar to the notion of block of $P(Q)$-bridges.)

Notice that all bridges that contain these cross-cuts have their only vertex of attachment as b. Let p_s and p_t be the two end vertices of a cross-cut in CC_P. Let $V' = CC_P \cup \{P[p_s, p_t]\}$. Let v_s and v_t be two end vertices in V' which are respectively nearest to s and t in the path $D[s, t]$. Let v_u and v_v be similarly defined but with respect to $D[u, v]$. Notice v_u and v_v are distinct and all the four vertices are in V'. Moreover, one of these four vertices is in $P]p_s, p_t[$ and is different from a (otherwise there is no path from a vertex of V' to a vertex in Q). Let this vertex be v_i. This vertex is incident with a PQ-cross-cut $CC_1[v_i, v_q]$, where v_q is a vertex in Q. Otherwise v_i or v_q is not in $P[p_s, p_t]$ which violates the property of p_s or p_t or, v_i is not one of v_s, v_t, v_u, v_v which is a contradiction. Further, it is easy to see v_t is different from b. It is easy to see that this cross-cut is in CC_{PQ}. Therefore, there is another cross-cut in CC_{PQ}, $CC_2[x, y]$ that interlaces with this cross-cut. It is easy to see that either x or y together with v_q satisfy condition (1) – a contradiction.

In the next section we give an algorithm for TPP on G. This is essentially to summarize the cases considered so far. □

3.3. Algorithm for TPP

INPUT: A biconnected planar graph G.
OUTPUT: A Boolean flag TPP which is set to TRUE iff there exists two disjoint paths $D[s, t]$ and $D[u, v]$ and the two paths if they exist.
MODEL: CRCW PRAM.
METHOD:
(1) Find an s–t ambitus C and all the bridges with respect to C.
(2) IF there is an s–t bridge THEN,
  (2a) IF observation (1) is satisfied THEN,
    (2a1) return (TPP:= TRUE). Two paths are constructed as given there.
    (2a2) ELSE return (TPP:= FALSE).

(3) ELSE
(4) IF $u(v)$ is in the nucleus of a PQ-bridge THEN,
  (4a) return (TPP:= TRUE). Construct paths as given in cases (1a) and (1b) of Section 3.2.
(5) IF $u(v)$ is in nucleus of a P-bridge.
  (5a) IF $v(u)$ is in the nucleus of a P-bridge THEN,
    (5a1) return (TPP:= TRUE). Construct paths as given in case (2a) of Section 3.2.
  (5b) IF $v(u)$ is in the nucleus of a Q-bridge THEN
    (5b1) IF $v(u)$ satisfy conditions given in Theorem 1 THEN construct the paths as in the proof of the Theorem 1.
    (5b2) ELSE return (TPP:= FALSE).
(6) END.

Proof of correctness. Obvious from the above lemmas. □

Before we proceed with complexity analysis, we make a couple of observations which make the analysis simple.

Observation 2. *Let a, b, c, d be four vertices such that they are as given in Lemma 4. Let there be a PQ-cross-cut from a to d using the bridge B_1 and another from b to c using B_2. Then there is a cross-cut between $s_P(B_1)$ and $t_Q(B_1)$ using B_1, and another between $t_P(B_2)$ and $s_Q(B_2)$ using B_2.*

Proof. Obvious. □

Observation 3. *Let a, b, c, d be as in Observation 2. Then there is no vertex e, strictly to the left of a that can cross-over with a.*

Proof. Let f be a vertex to the right of d. Notice the cycle $CC_1[a, d] * Q[d, c] * CC_2[c, b] * P[b, a]$, where CC_1 and CC_2 are, respectively, cross-cuts through B_1 and B_2, is a Jordan curve. This curve will partition the plane into two regions one containing e and another containing f. Thus, by Jordan curve theorem, if there is a cross-cut from e to f then, the graph is not planar. □

Complexity analysis. For each bridge B, maintain its label (indicate whether it is P or Q or PQ-bridge), its vertices, edges, all vertices of attachment and the four vertices $s_{P(Q)}(B)$, $t_{P(Q)}(B)$. Without loss of generality we assume that the vertices in P and Q are stored in two different arrays. With this, it can be easily seen that all the steps, except step (5b1), in the algorithm can be easily implemented in $O(\log n)$ time using $O(n/\log n)$ processors. Now, we show how to implement the step (5b1) within the processor-time bound claimed. For that do the following.

Step 1: Let B be a bridge. If $s_P(B)$ is to the left of u and $t_Q(B)$ is to the right of v then mark $s_P(B)$ and $t_Q(B)$. Rank all marked vertices in P and denote the rank by $R_P(x)$, in the increasing order, as they occur in P from s to t. Do similar ranking to all the marked vertices in Q and denote it by $R_P(y)$ for any marked vertex y. The rank of a cross-cut, using the bridge B, is a pair $R_P(s_P(B)), R_Q(t_Q(B)))$. If the ranks in the two coordinates are the same for all cross-cuts, then mark all vertices $t_P(B')$ and $s_Q(B')$ and unmark corresponding $s_P(B')$ and $t_Q(B')$ and do similar ranking for all the marked vertices.

Step 2: Declare a cross-cut $CC[s_P(B), t_Q(B)]$ to be one of the two cross-cuts satisfying condition (1) of Theorem 1 iff the two coordinates in the rank of a cross-cut are different.

We claim that this procedure detects one of the two cuts satisfying condition (1) of Theorem 1, if they exist. To prove this notice that, by Observation 2, for any bridge, it is sufficient to maintain (only) the vertices which are nearest to s and t in P and Q. Let B and B' be any two arbitrary bridges such that the cross-cuts $CC[s_P(B), t_Q(B)]$ and $CC[x, y]$ use B and B', respectively. Further, let the two cross-cuts satisfy condition (1) of Theorem 1 and $s_P(B)$ is to the left of u and $t_Q(B)$ to the right of v.

(a) If $t_P(B')$ is strictly to the right of $s_P(B)$ then, $R_Q(t_q(B)) > R_P(s_P(B))$ else,

(b) if $t_P(B')$ is strictly to the left of $s_P(B)$ then, $R_Q(t_Q(B)) < R_P(s_P(B))$.

In fact the two coordinates in the rank of the cross-cut never differ by more than 1.

Since there cannot be two cross-cuts (by Observation 3), one satisfying case (a) and another satisfying case (b), the step 2 in the above procedure correctly declares the one of the two cross-cuts, satisfying condition (1) of Theorem 1. Condition (1a) can be similarly checked. Step 1 can be implemented as follows. If a vertex is marked give it a value 1 else, give it 0. Now find prefix sums on these values. The prefix sum at each marked vertex gives its rank. Since prefix sums can be found in $O(\log n)$ time using $O(n/\log n)$ processors [15], steps 1 and 2 can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors. Also, the cross-cut interlacing with CC can be found in $O(\log n)$ time using $O(n/\log n)$ processors using standard techniques. Notice that a vertex can be under at most two bridges in a planar graph. Hence, condition (3) can be checked in $O(\log n)$ time using $O(n/\log n)$ processors. Also, if any one of the conditions in Theorem 1 is satisfied, then the paths $D[s, t]$ and $D[u, v]$, as given in the proof of Theorem 1 can be easily constructed as follows: the algorithm in [13] restricted to planar graphs will yield the disjoint paths in $O(\log n)$ time using $O(n/\log n)$ processors because of the results in [6]. We will also use the fact that the spanning tree of a planar graph can be constructed optimally in $O(\log n)$ time, using $(n/\log n)$ processors. Hence, we have the following theorem.

Theorem 2. *The procedure TPP sets TPP to TRUE iff both* $D[s, t]$ *and* $D[u, v]$ *exist and finds them whenever they exist in* $O(\log n)$ *time. The processor count is bounded by that required for integer sorting in* $O(\log n)$ *time on a CRCW PRAM.* □

4. Algorithmic details of all-bidirectional-edges problem

In this section, we show how to implement ABE problem on planar graphs as given in [19]. We will not give the details of the algorithm as they are available elsewhere [19].

Lemma 5. *A planar graph can be divided into triconnected components in* $O(\log n)$ *time. The number of processors is limited by that required to perform integer sorting in* $O(\log n)$ *time on a CRCW PRAM.*

Proof. From the results in [4–6] it is easy to see that this bound can be achieved. □

In [19] they reduce the ABE problem on G to the one on its triconnected components. The key subproblem is the case when both s and t are in a single triconnected component; other cases can be trivially handled in parallel. To solve this subproblem we have to solve the following problems:

Problem 1. Contract the subpath between some marked pairs of vertices in P or Q.

Problem 2. In a cycle C, report all the edges which are under interlacing bridges. We give parallel implementation for these problems below.

Parallel implementation.

Problem 1. Let there be k vertices ($k = O(n)$). Without loss of generality we assume that the vertices are stored in a linked list. Create a field for every vertex and set it to 1 if it is marked else set it to zero. Rank the vertices in the list using any list ranking algorithm and find prefix sums on these values. Now, if prefix sums at ranks i and $i+1$, $1 \leqslant i \leqslant k-1$ are same, then make vertex at rank i point to the vertex at rank $i+2$. This recursively doubles at each step and thus can be implemented in $O(\log k)$ time using $O(k/\log k)$ processors.

Problem 2. We mark all the vertices which are vertices of attachment of at least one bridge and order these vertices in a cyclic order, as they occur in C in clockwise order and store them in a circular list. This can be done by Problem 1. With each vertex v, in the circular list, we allot at most degree number of processors each storing a neighboring vertex in each of the bridges in which v is contained. Of these vertices find the vertices with maximum and minimum value, say respectively max_v and min_v and store them in the processor allotted to v. Now each processor looks at both of its neighbors in the circular list, say u and w and find if $CC(v, min_v)$ interlaces with $CC(u, max_u)$ or $CC(w, max_w)$ (or both), and mark all the vertices which are under the interlacing cross-cuts. It can be easily seen that this marking marks a vertex iff it is under

interlacing bridges. It can be seen that this can be implemented in $O(\log n)$ time with $O(n/\log n)$ processors using standard recursive doubling technique.

Theorem 3. *ABE problem can be solved on a biconnected planar graph in $O(\log n)$ time using $O(n/\log n)$ processors on a CRCW PRAM.*

5. Summary

In this paper we gave simple optimal algorithm, running in $O(\log n)$ time for two path problem, on planar graphs without dividing the graph into triconnected components. Our algorithm exploits many interesting structural properties of bridges of biconnected planar graphs. We have also outlined an optimal parallel implementation of the algorithm presented in [19], running in $O(\log n)$ time, for ABE problem on planar graphs.

6. Conclusions and open problems

It is interesting to see if, extending this method, Kurtowski's homeomorph can be detected in case the graph is not planar. The best solution known for finding Kurtowski's homeomorph is nonoptimal [10]. Also, it is interesting to see if this method can be extended in conjunction with connectivity algorithms to find disjoint paths between k, $k > 2$, pairs of vertices. We are presently working in this direction.

Acknowledgements

We would like to thank the anonymous referee for numerous suggestions which helped in improving the style of the paper. We would like to thank Dr. Hagerup, Max Plank Institute for Informatics, Germany, for several useful discussions during the development of this paper.

References

[1] R.J. Anderson and G.L. Miller, Deterministic parallel list ranking, in: *Proc. AWOC*, Lecture Notes in Computer Science, Vol. 319 (Springer, Berlin, 1988) 81–90.

[2] J.A. Bondy and U.S.R. Murthy, *Graph Theory with Applications* (North-Holland, New York, Oxford, 1979).

[3] S. Fortune, J.E. Hopcroft and J. Wyllie, Directed sub graph homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980) 111–121.

[4] D. Fussel, V. Ramachandran and R. Thurimella, Finding triconnected components by local replacements, in: *Proc. 16th ICALP*, Lecture Notes in Computer Science, Vol. 372 (Springer, Berlin, 1989) 379–393.
[5] T. Hagerup, Towards optimal parallel bucket sorting, *Inform. and Comput.* **75** (1975) 39–51.
[6] T. Hagerup, Optimal parallel algorithms on planar graphs, in: *Proc. AWOC*, Lecture Notes in Computer Science, Vol. 319 (Springer, Berlin, 1988) 24–32.
[7] J. Jaja, *An Introduction to Parallel Algorithms*, (Addison-Wesley, Reading, MA, 1992).
[8] R.M. Karp, On computational complexity of combinatorial problems, *Networks* **5** (1975) 45–68.
[9] B. Korte, L. Lavsz, H.S. Prmel and A. Schrijver, eds., Paths, flows and VLSI-layout, *Algorithms Combin.* **9** (1990) 266–292.
[10] S. Khuller, S.G. Mitchell and V.V. Vazirani, Processor efficient algorithms for the two disjoint paths problem and for finding a kurtowski's homeomorph, in: *Proc. 30th FOCS* (1989) 300–305.
[11] K.S. Easwarakumar, S.V. Krishnan, C. Pandu rangan and S. Seshadri, Optimal parallel algorithm for finding *st*-ambitus of a planar biconnected graph, *Algorithmica*, to appear.
[12] R.M. Karp and V. Ramachandran, Parallel algorithms for shared memory machines, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (Elsevier, Amsterdam, 1990) 869–941.
[13] S. Khuller and B. Schieber, Efficient parallel algorithms for testing connectivity and finding disjoint s–t paths, in: *Proc. 30th FOCS* (1989) 288–293.
[14] J.F. Lynch, The equivalence of theorem proving and interconnection problem, *ACM SIGDA Newslett.* **5** (1975) 31–65.
[15] R.E. Ladner and M.J. Fischer, Parallel prefix computation, *J. ACM* **27** (1980) 831–838.
[16] B. Mishra, Graph theoretic issues in VLSI design, Ph.D. Thesis Carnegie–Mellon University.
[17] T. Ohtsuki, Two disjoint path problem and wire routing design, *Graph Theory and Applications, Vol. 108* (Springer, Berlin, 1980) 207–216.
[18] Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978) 1–9.
[19] P.B. Ramprasad and C. Pandu Rangan, A linear algorithm for all-bidirectional-edges problem on planar graphs, *Algorithmica* **9** (1993) 199–216.
[20] V. Ramachandran and J. Reif, Optimal parallel algorithm for graph planarity, in: *Proc. 30th FOCS* (1989) 282–287.
[21] P.D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980) 293–309.
[22] Y. Shiloach, A polynomial solution to the undirected two path problem, *J. ACM* **27** (1980) 445–456.
[23] R.E. Tarjan and U. Vishkin, An efficient biconnectivity algorithm, *SIAM J. Comput.* **14** (1984) 862–874.
[24] W.T. Tutte, *Graph Theory* (Addison-Wesley, Menlo Park, CA, 1984).